# SegmentStyler: Part-aware language-based mesh texture editing

Maximilian Winter
maximilian96.winter@tum.de

Murilo Bellatini
bellatini@in.tum.de

## Abstract

*This study proposes a novel framework for part-aware mesh texture editing based on natural language. Given an uncolored input mesh, the framework predicts vertex colors which conform a set of target text prompts. It leverages the powerful zero-shot capabilities of CLIP and is build upon Text2Mesh. It further allows separate customization of different segments of the mesh (e.g., arms, legs, seat, back of a chair) by using the language-based part segmentation predictions of PartGlot. We show that the framework can be successfully used for part-aware texturing of chairs, outperforming the baseline method by a significant margin. These results indicate high potential for part-aware mesh editing beyond the furniture domain. Our code is available at* https://github.com/MaximilianWinter/SegmentStyler.

## 1. Introduction

Recent advancements in the field of neural image generation, specifically in the development of advanced tools for creating and manipulating images based on natural language, have sparked significant interest in academia and industry. Notable examples of such frameworks include StableDiffusion [1] and DALL-E [2]. Similarly, the field of 3D computer vision has also followed the same trend, with advances in language-based shape generation such as CLIP-Mesh [3], CLIP-Forge [4] or GET3D [5]. Mesh editing also provided innovation with the emergence of Text2Mesh [6], a framework capable of stylizing a given mesh by predicting color and local geometric details based on a text prompt.

Despite the promising results of existing frameworks, the current methods often still fall short of manipulating the mesh editing process with fine-grained details, in particular with different textures for individual parts of the mesh. This is especially apparent for the manipulation of general object shapes (like furniture) with CLIP-based methods, as CLIP [7] has limited knowledge on different object parts. This gap in the field motivated us to pursue a framework for mesh editing with part-awareness. Specifically, we propose *SegmentStyler*, a novel approach for editing the vertex colors of 3D objects based on a set of text prompts. Our framework builds upon the color editing part of the existing method Text2Mesh [6], but introduces the novel capability of editing 3D objects differently for each of its segments. This means that users can now specify different color patterns for multiple parts of the input mesh at once.

This is achieved by first generating part maps for each of the given text prompts using a pre-trained version of PartGlot [8]. These part maps highlight all object parts mentioned in the specific prompt. For each text prompt, we train a MLP to predict the vertex colors of the specified parts by maximizing the cosine similarity of the text prompt's CLIP embedding and the embeddings of the stylized 3D shape's renderings. We use the part maps to make sure that each MLP only receives gradients from its corresponding semantic loss. Additionally, we employ various strategies, such as biasing the renderings' viewpoints or applying the label propagation algorithm [9] to the part maps, to further improve the result. We thereby clearly outperform Text2Mesh for our control task (i.e. styling chairs' segments).

The main contributions of this study can be summarized as follows:

- The presentation of a novel framework for editing the texture of 3D objects based on a set of part-specific text prompts, using multiple combined strategies.
- The introduction of the capability to edit 3D objects differently for each of their segments, with preliminary results that are promising for further investigation.

## 2. Related Work

There are many existing frameworks for generating [3, 4, 10, 11] or editing [6, 12, 13] images or 3D shapes based on CLIP [7]. One of them, Text2Mesh [6], forms the cornerstone of our implementation, together with PartGlot [8]. Due to their importance for our work, we will summarize those methods in the following.

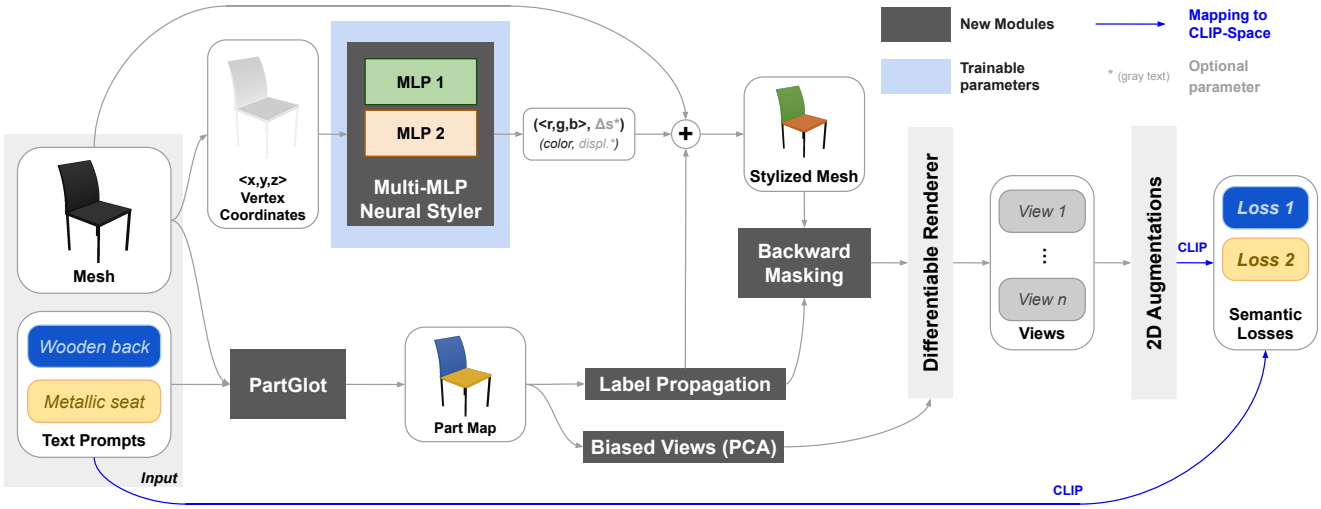


Figure 1. Our architecture is build upon Text2Mesh [6]. In order to clearly visualize our contributions, we indicate them by dark gray boxes. Our pipeline takes as input a mesh and a set of $k$ text prompts, each corresponding to a single segment of the input mesh. In this visualization $k$ is set to 2. The main goal is to produce a stylized mesh that is coherent with the provided text prompts. To achieve this, we use a semantic loss in CLIP-space for each prompt. The text prompts are directly mapped into CLIP-space, while the mesh undergoes a more complex pipeline. First, for each vertex we get a semantic part label from **PartGlot**. Next, the vertices are processed by the **Multi-MLP Neural Styler**, which predicts color and optional displacement values (not used) for each vertex. The stylized points are then combined with the input mesh to produce a stylized mesh that is coherent with the provided text prompts. The **Backward Masking** module uses PartGlot's part maps to mask the gradient flow during backpropagation to ensure that each MLP only receives gradients from its corresponding loss. The resulting stylized mesh is rendered into multiple views, which are guided by the **Biased Views** module to ensure that all relevant object parts are well captured. The rendered images are then augmented and mapped into CLIP-space to compute the loss for each prompt-part pair. Additionally, our **Label Propagation** module is used to ensure that the part maps have a more consistent behavior, eliminating empty patches and leading to a more coherent stylized mesh.

### 2.1. Text2Mesh: Mesh Editing

Text2Mesh is a framework for editing meshes using natural language prompts. It takes a mesh and a text prompt as input, and produces a stylized version of the mesh. This framework was crucial to our project, as it forms the basis of our approach to mesh editing. We adapted the Text2Mesh repository[1] to suit our needs.

The Text2Mesh approach uses a multi-layer perceptron (MLP) which predicts vertex color and displacements to create the stylized mesh and is optimized based on a CLIP-space loss. To accomplish this, multiple views of the stylized mesh are rendered, and the CLIP embedding of each view is extracted. A similar process is applied to the text prompt, which is directly mapped into a CLIP embedding. The loss is calculated based on the cosine similarity of these two sets of CLIP embeddings.

The main limitation of Text2Mesh is that it only allows for a single mesh-text prompt pair as input. In our study, we aimed to address this limitation and make it possible to use multiple text prompts, each referring to a different segment of the mesh. Note, however, that for the sake of simplicity we do not utilize the vertex displacements in this work and only take the predicted vertex colors into account.

[1] https://github.com/threedle/text2mesh

### 2.2. PartGlot: Part Segmentation

PartGlot is a neural framework that allows for the semantic part segmentation of 3D shapes. To achieve this, PartGlot leverages natural language descriptions as prior information, exploiting the way humans perceive the compositional structure of objects. Whereas PartGlot's novelty lies in the way its training works, its importance for our project comes from the fact that it predicts the corresponding part of a given shape based on a natural language input.

To integrate PartGlot into our pipeline, we utilized the pre-trained weights provided by the authors and modified their code to extract the part maps in runtime.

## 3. Method

Given an input mesh $\mathcal{M}_{\text{in}}$ with vertices $V \in \mathbb{R}^{N\times 3}$ and $k$ text prompts $\{t_i\}_{i=1}^{k}$, we first infer for each prompt a binary part map $m_i \in \{0,1\}^N$ using PartGlot. We stack the one-dimensional maps to get $\tilde{M}_i = (m_i, m_i, m_i) \in \mathbb{R}^{N\times 3}$ and improve certain imperfections by applying the label propagation algorithm, leading to $M_i$. We additionally have $k$ different MLPs ("neural stylers") $\{f_{\theta_i}\}_{i=1}^{k}$. Each $f_{\theta_i}$ takes

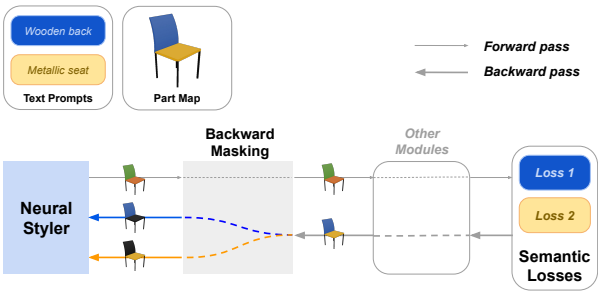


Figure 2. The **Backward Masking** module ensures that each MLP only receives gradients from its corresponding loss, by acting as identity element during the forward pass and masking out components of the incoming gradient tensor during the backward pass. See fig. 1 for its utilization in the overall pipeline.

the vertex positions $V$ and maps them to RGB vertex colors $C_i = f_{\theta_i}(V) \in \mathbb{R}^{N\times 3}$. We aggregate the different color predictions to one color tensor via $C = \sum_{i=1}^{k} C_i \odot M_i$. Subsequently, we create $k$ copies of $C$ which all behave differently during the backward pass. To emphasize this in our notation, we denote such tensors that only differ in their backward behavior by an upper index, i.e. $C^i = f_{\text{bm},i}(C)$. The $C^i$ are then combined with $\mathcal{M}_{\text{in}}$ to produce the *stylized* meshes $\mathcal{M}^i_{\text{out}}$. We subsequently create renderings of each stylized mesh from multiple view points using a differentiable renderer, perform certain augmentations on the resulting images and embed them using CLIP. These three steps, which we denote as $\sigma_i = g_i(\mathcal{M}^i_{\text{out}})$, are largely equivalent to the corresponding steps in ref. [6]. Note, however, that the index $i$ indicates that this is done per text prompt. Specifically, we bias the renderings' view points based on the part maps $\tilde{M}_i$, as explained in more detail below. We also map each text prompt $t_i$ to CLIP space, which we denote as $\tau_i$, and formulate its corresponding semantic loss $\mathcal{L}_i$ as the negative cosine similarity between $\tau_i$ and $\sigma_i$.[2] The full loss is given as $\mathcal{L} = \sum_{i=1}^{k} \mathcal{L}_i$. A high-level overview of our method is visualized in fig. 1 for $k = 2$.

### 3.1. Backward Masking

Backward masking is a technique used to encourage weight updates for isolated part-prompt pairs. This is achieved by using (PartGlot's) part maps to mask the gradient flow during backpropagation. A schematic of this idea is shown in fig. 2.

For an input tensor $C \in \mathbb{R}^{N\times 3}$ and a binary mask $M_i \in \mathbb{R}^{N\times 3}$ as defined above, we define the backward masking layer by the following properties:

$$f_{\text{bm},i}(C) = C =: C^i$$
$$\frac{\partial f_{\text{bm},i}(C)}{\partial C} = M_i$$

This means that the layer acts as identity element during the forward pass, while masking out components of the incoming gradient tensor during the backward pass. We can thereby ensure that each MLP with weights $\theta_l$ only receives gradients from its corresponding semantic loss $\mathcal{L}_l$, while performing a single backward pass on the full loss $\mathcal{L}$ (and thereby on all MLPs simultaneously). This can be shown mathematically as follows:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \sum_{i=1}^{k} \frac{\partial \mathcal{L}_i}{\partial C} \odot \underbrace{\frac{\partial C^i}{\partial C}}_{=M_i} \odot \sum_{j=1}^{k} M_j \odot \frac{\partial C_j}{\partial \theta_l} \tag{1}$$

$$= \sum_{i=1}^{k} \frac{\partial \mathcal{L}_i}{\partial C} \odot \underbrace{M_i \odot M_l}_{=\delta_{il} M_l} \odot \frac{\partial C_l}{\partial \theta_l} \tag{2}$$

$$= \frac{\partial \mathcal{L}_l}{\partial C} \odot \frac{\partial \left(C_l \odot M_l\right)}{\partial \theta_l} \tag{3}$$

Here we used the fact that the part assignments are unique. The symbol $\delta_{il}$ in eq. 2 denotes the Kronecker delta. It is important to note here that even for a single MLP (i.e. $C = C_1$) this layer can help to achieve distinct styles between parts. This is because masking out the gradients for certain vertices has a similar effect as not considering these vertices at all for the MLP's weight update. Thus, effectively, the optimization for each semantic loss is solely done with respect to vertices (= "samples") that represent the corresponding part.

### 3.2. Biased Views

When training our model using only the methods explained so far, we observed that some of the MLPs, especially the ones corresponding to small parts, often do not converge. The presumed reason for this observation is that small parts are not well captured in global renderings of the shape and thus barely have any gradients associated with them during the backward pass.

In order to improve this issue, we bias the renderings' view points for each text prompt based on its associated part map. Specifically, we take the segmented vertex positions $V_i \in \mathbb{R}^{N_i\times 3}$ with $N_i \leq N$ and apply principal component analysis (PCA) to determine the three principal axes of the segmented part. For homogeneously sampled vertices, this is equivalent to calculating the eigenvectors of the part's moment of inertia tensor. The subsequently created view points are sampled on a sphere centered at the part's center of mass with a radius proportional to the part's

[2] We refer to [6] for details on the aggregation of the renderings' embeddings.

extension. For the sampling we use two independent, one-dimensional normal distributions for the azimuth angle and the elevation, both centered along the (randomly flipped) principal axis with the smallest principal value. This effectively leads to a large projection surface of the part in the resulting renderings.

### 3.3. Gaussian Blending

One possibility to mitigate the impact of imperfect part maps coming from PartGlot, is to approximate them by Gaussians. For this we calculate the center of mass and covariance matrix of a part's vertices $V_i$, and use them as parameters for a Gaussian $\mathcal{G}(\mu_i, \Sigma_i)$.[3] The new part maps are then calculated as the Gaussians' value at the vertex positions, i.e. $m_{\mathcal{G},i} = \mathcal{G}(\mu_i, \Sigma_i)(V) \in \mathbb{R}^N$. We additionally normalize them such that for each vertex $j$ we have $\sum_{i=1}^{k} m_{\mathcal{G},ij} = 1$. Note that the resulting part maps are not binary anymore, but lie in the range $[0, 1]$.

This technique can fix certain inconsistencies in PartGlot's predictions, such as holes, and can be used instead of Label Propagation in our pipeline. However, as not every part can be well approximated by a Gaussian and as there is no learned component in this approach, it does not always lead to an improvement in the resulting part maps. Therefore it is not used by default.

### 3.4. Label Propagation

Label Propagation [9] is a semi-supervised learning algorithm for labeling graphs. Given a graph with adjacency matrix $A$ and some labeled nodes, the goal is to predict the remaining nodes' labels $y_i \in \{0, 1\}$. This is done by minimizing the energy

$$E(y) = \frac{1}{2} \sum_{ij} A_{ij}(y_i - y_j)^2 \tag{4}$$

for the remaining nodes' labels. This energy decreases when neighboring vertices have the same label, thereby encouraging smoothness and discouraging label switches. As these are desired properties for our part maps, we can interpret the mesh as a graph and apply this algorithm to it.[4] Unfortunately, PartGlot's prediction scores cannot be reliably used for identifying "bad" labels. Therefore, instead of using a semi-supervised approach, we use PartGlot's labels as initialization and optimize for *all* of them until the energy reaches a certain threshold. To enable efficient optimization using gradient descent, we relax the discrete constraint on the labels. Note that this optimization is done independently from training the MLPs. Thus it can be either used as pre- or as post-processing step. Fig. 3 shows an example of PartGlot's label prediction and the corresponding corrections using Gaussian Blending and Label Propagation.

[3] Note that for the case of a chair's arms (legs) we subdivide the given part map into 2 (4) sub parts using k-means clustering.

[4] The $K$-label version of the energy in eq. 4 can be written as $E(Y) = \sum_i \tilde{y}_i^T L \tilde{y}_i$ with $L$ being the graph Laplacian and $Y = (\tilde{y}_1, ..., \tilde{y}_K) \in \{0, 1\}^{N \times K}$ being the labels in one-hot-encoding.

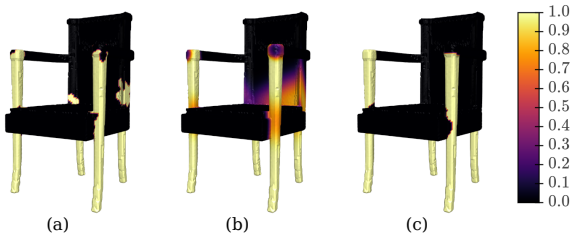


Figure 3. Comparison of PartGlot's prediction for "legs" (a), its Gaussian approximation using Gaussian Blending (b) and the result after performing Label Propagation (c). In this example, Label Propagation clearly leads to a significant improvement over (a) and (b), as there are no parts of the chair's back being erroneously labeled as "legs" anymore.

## 4. Results

In this section, we discuss the results of our method for a set of pre-defined text prompts and shapes. As we are using pre-trained weights for PartGlot[5], which was trained on chairs of the ShapeNet [14] dataset, we limit our selection of shapes to chairs too. Note, however, that the chairs selected for our evaluation were not seen during PartGlot's training.

We train our model using the pre-trained CLIP ViT-B/32 model and evaluate it using the more powerful CLIP ViT-L/14@336px architecture. We use 25 different meshes and 10 distinct sets of four prompts ($k = 4$), each referring to one of the four parts *arm, leg, seat and back*, resulting in a total of 40 distinct prompts $\{t_i\}_{i=1}^{40}$ and 250 different optimizations. A list of all used prompts and ShapeNet ids can be found in our code's repository. The architecture of our MLPs are the same as in [6]. However, we decrease the networks' width from 256 to 32 and the "shared" layers' depth from 4 to 2. We optimize the MLPs' weights for 1500 iterations, using the Adam optimizer [15] with an initial learning rate of $5 \times 10^{-3}$ and use the same learning rate schedule and the same augmentations as in [6].

### 4.1. Pre-processing

As many meshes in ShapeNet have incorrect normals, which are required for getting correct shading in the renderings, we remesh them before applying our pipeline. For this we use the `mesh_to_sdf` package[6], which makes sure that all normals point outwards by performing virtual scans, together with the marching cubes algorithm. This has the

[5] Due to its better performance we use the PN-aware model.

[6] https://github.com/marian42/mesh_to_sdf

| Method | R-Precision (R=1) |
|---|---|
| Baseline (Text2Mesh) | 0.376 |
| Ours (SegmentStyler) | **0.612** |

Table 1. Quantitative comparison of our work with Text2Mesh for 250 distinct shape-prompt-pairs. Our method clearly outperforms the baseline by a large margin.

convenient side effect of resulting in a more regular meshing, meaning the stylization has a similar resolution at each section of the mesh. Additionally, the pre-processing returns watertight meshes in most cases (and specifically for the shapes we selected), which makes the label propagation algorithm more powerful.

### 4.2. Quantitative Evaluation

For evaluating our method, we use a variation of CLIP R-Precision, which was also used in [3] and was first described in [16]. This metric measures the top-$R$ retrieval accuracy when retrieving the matching text from a set of text candidates using the stylized mesh as a query. As matching criterion we use the cosine similarity in CLIP space. Specifically, for any text prompt $t_i$ we compute its cosine similarity in CLIP space with a rendering of the stylized mesh from a fixed view point. To improve the metrics' fidelity and to account for the fact that CLIP has no good understanding of different furniture parts, we only render the part that is mentioned in $t_i$. For this segmentation, we use the ground truth part labels. We count the retrieval as successful if any of the four utilized prompts $t_i^*$ is among the top-$R$ scoring prompts. In the following evaluation, we set $R = 1$.

We conduct the same evaluation with Text2Mesh, to be able to assess our method's performance in comparison. Note, however, that Text2Mesh requires a single text prompt for training, which we create by simply appending the prompts $\{t_i^*\}_{i=1}^4$ using commas. We additionally add the prefix "a chair with a" to the resulting prompt.

The quantitative result of this evaluation on all 250 stylized meshes is shown in tab. 1. Our method outperforms Text2Mesh by a large margin.

### 4.3. Ablation Study

To assess the effectiveness of each component in our architecture, we conduct an ablation study by starting off with the baseline and cumulatively adding the various features explained in sec. 3. Note, however, that this analysis only involves 219 stylized meshes ($219/250 = 87.6\%$) per stage. The quantitative results of this study are shown in tab. 2. Additionally, we show some qualitative examples in fig. 4.

Each feature we add leads to some improvement, both quantitatively as well as qualitatively. Specifically, we observe that the baseline method in a) struggles to distinguish

| Ablation study stage | R-Precision (R=1) |
|---|---|
| a) Baseline | 0.370 |
| b) + Backward Masking | 0.447 |
| c) + Multi MLPs | 0.520 |
| d) + Biased views | 0.612 |
| e) + Blending | 0.617 |
| f) + Label Propagation, - Blending | 0.625 |
| g) GT masks | 0.630 |

Table 2. Ablation study for 219 distinct shape-prompt-pairs, cumulatively adding components from top to bottom. In stage f), Label Propagation was applied as post-processing to the pre-trained models from d). For comparison, we used the ground truth masks instead of relying on PartGlot's predictions in stage g).

the different parts of the shape and often globally converges to one of the styles mentioned in the prompt. This gets clearly improved by adding backward masking (and multiple semantic losses) in b), as it effectively creates some level of part awareness in the model. Introducing multiple MLPs in c) creates further separation between the different parts, whereas biasing the renderings' view points in d) helps the MLPs to converge, as explained in sec. 3.2. In the subsequent stages e) and f) we try to account for imperfect part maps, which leads to marginal quantitative improvements. It is important to note here, that for f) we did not train the model from scratch, but instead used the pre-trained model from d) and applied label propagation as post-processing. For the last stage g) we use the ground truth part labels instead of PartGlot's predictions for comparison.

## 5. Conclusion

Our proposed approach for text-driven 3D mesh editing demonstrates promising results and significantly outperforms the baseline (Text2Mesh) in the task of mesh editing with part-awareness, showing an improvement of about $63\%$ in R-Precision with $R = 1$. While our experiments focused on the chair domain, our framework has the potential to be extended to other domains as well. The current limitation lies in the part segmentation model. However, to enable extension to other shape categories, PartGlot could be retrained for other domains. Additionally, our framework is versatile enough to work with vertex displacements in addition to color, further increasing its potential applications. Although the former was turned off during evaluation to reduce complexity, future work could assess the predicted displacements to create a detailed 3D texture. To further improve the results one could also incorporate symmetry constraints to the optimization or fine-tune CLIP to improve its understanding of object parts.

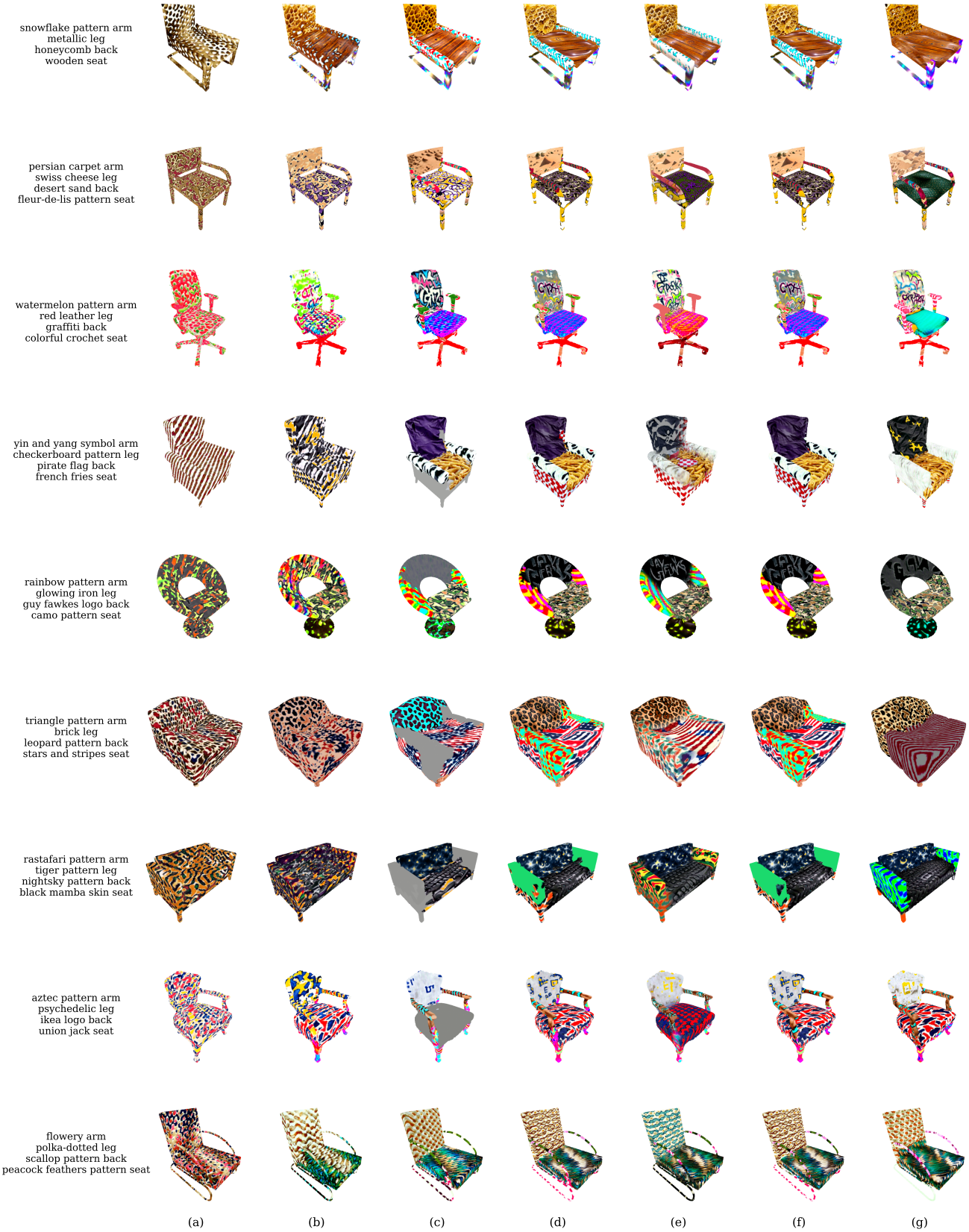


Figure 4. Some examples for the different stages of the ablation study, indicated by (a)-(g). The performance clearly improves from left to right. For a more detailed discussion, see the main text.

## References


[1] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 1

[2] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *CoRR*, abs/2102.12092, 2021. 1

[3] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. CLIP-mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*. ACM, nov 2022. 1, 5

[4] Aditya Sanghi, Hang Chu, Joseph G Lambourne, Ye Wang, Chin-Yi Cheng, and Marco Fumero. Clip-forge: Towards zero-shot text-to-shape generation. *arXiv preprint arXiv:2110.02624*, 2021. 1

[5] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, 2022. 1

[6] Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. Text2mesh: Text-driven neural stylization for meshes. *CoRR*, abs/2112.03221, 2021. 1, 2, 3, 4

[7] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. 1

[8] Juil Koo, Ian Huang, Panos Achlioptas, Leonidas J. Guibas, and Minhyuk Sung. Partglot: Learning shape part segmentation from language reference games. *CoRR*, abs/2112.06390, 2021. 1

[9] Xiaojin Zhu and Zoubin Ghahramani. Learning from Labeled and Unlabeled Data with Label Propagation. 2002. 1, 4

[10] Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. Zero-Shot Text-Guided Object Generation with Dream Fields, May 2022. arXiv:2112.01455 [cs]. 1

[11] Katherine Crowson, Stella Biderman, Daniel Kornis, Dashiell Stander, Eric Hallahan, Louis Castricato, and Edward Raff. VQGAN-CLIP: Open Domain Image Generation and Editing with Natural Language Guidance, September 2022. arXiv:2204.08583 [cs]. 1

[12] Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery, March 2021. arXiv:2103.17249 [cs]. 1

[13] Yongwei Chen, Rui Chen, Jiabao Lei, Yabin Zhang, and Kui Jia. Tango: Text-driven photorealistic and robust 3d stylization via lighting decomposition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 1

[14] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository, December 2015. arXiv:1512.03012 [cs]. 4

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 4

[16] Dong Huk Park, Samaneh Azadi, Xihui Liu, Trevor Darrell, and Anna Rohrbach. Benchmark for Compositional Text-to-Image Synthesis. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. 5